Universitat de Lleida

TREBALL FINAL DE MÀSTER

Estudiant: **José David Nunes de Oliveira**

Titulació: Màster en Enginyeria Informàtica

Títol de Treball Final de Màster: Aplicación web de apoyo al regante

Director/a: Jaume Casadesús, Toni Granollers

Presentació

Mes: Setembre

Any: 2018

José David Nunes de Oliveira

APLICACIÓN WEB DE APOYO AL REGANTE

Trabajo de Final de Máster presentado
a la universidad de Lleida.

Orientadores: Prof. Dr. Toni Granollers,
Dr. Jaume Casadesus

Lleida/ES

2018

A mis padres, que me educaron para llevar una vida digna
en que yo pudiese crecer, creyendo que todo es posible,
partiendo de la honestidad.

AGRADECIMIENTOS

Agradezco a Dios por darme fuerza y salud. A mis padres y a mi familia por confiar en mí y enseñarme sobre la vida. A la universidad FACENS de Brasil por ofrecerme la oportunidad de estudiar un máster en el extranjero. A mis tutores Toni Granollers y Jaume Casadesús, a mi novia Carmen España, amigos, por el apoyo e aliento que me han infundido para seguir hasta finalizar este trabajo. A Jordí Mòdol por la revisión de la memoria.

El verdadero signo de la inteligencia
no es el conocimiento, sino la imaginación.
Albert Einstein

RESUMEN

Oliveira, J. D. N. **Aplicación web de apoyo al regante**. Lleida, 2018, 60. Trabajo final de Máster – Máster en ingeniería informática, Universidad de Lleida. Lleida, 2018.

Este proyecto tiene por objetivo rediseñar una aplicación web que integrará diversas funcionalidades de apoyo al regante. Como punto de partida, en el IRTA tienen diversas aplicaciones propias, desarrolladas ad hoc en distintos proyectos de investigación y plantean integrarlas en una sola aplicación que sea más usable, más escalable y más fácil de mantener. La propuesta es analizar la arquitectura que mejor se adapte a los requisitos y demostrar su funcionalidad con dos casos de uso, uno de entrada de datos y otro de salida de datos. La arquitectura de la aplicación servirá como base para el desarrollo de casos de usos más avanzados. A grandes rasgos, el usuario podrá acceder a la página web para entrar las características de cada sector de riego, analizarlos riegos realizados hasta la fecha, consultar la programación de riego prevista para los próximos días y modificarla si lo considera oportuno.

Palabras clave: Sistemas de riego, Agricultura de precisión, Python, Django, MVC, NoSQL, MongoDB.

ABSTRACT

Oliveira, J. D. N. **Aplicación web de apoyo al regante**. Lleida, 2018, 60. Trabajo final de Máster – Máster en ingeniería informática, Universidad de Lleida. Lleida, 2018.

This project aims to redesign a web application that will integrate various support functions to the irrigator. As a starting point, IRTA has several own applications, developed ad hoc in different research projects and intends to integrate them into a single application that is more usable, more scalable and easier to maintain. The proposal is to analyze the architecture that best suits the requirements and demonstrate its functionality with two use cases, one for data entry and another for data output. The architecture of the application will serve as a basis for the development of cases of more advanced uses, initially the user will be able to access the website with data from the irrigation area, consult analysis of future risks and modify the information of this data.

Key-words: Irrigation System, Precision farming, Python, Django, MVC, NoSQL, MongoDB.

LISTA DE FIGURAS

SUMARIO

1. OBJETIVOS

El objetivo principal de este proyecto es desarrollar un prototipo de aplicación web de apoyo al regante, en el cual los investigadores del IRTA se puedan basar en decidir varias cuestiones de diseño para la refactorización de sus aplicaciones. Concretamente, en el marco de trabajo en aplicaciones para agricultura de precisión se plantean los siguientes subobjetivos:

- Probar una arquitectura y framework de referencia para una aplicación web que interaccione con el regante y, por detrás, que sea fácil de integrar con las librerías de código específico del dominio desarrolladas por investigadores del IRTA.
- Probar una solución para el almacenamiento de datos poco estructurados, consistentes en toda la información disponible sobre cada finca.
- Probar una solución para el almacenamiento y manejo de series temporales de datos largas, densas y creciendo continuamente, que generan los sensores.
- Probar una solución práctica para la edición, almacenamiento y uso de datos geográficos en este tipo de aplicación.
- Consolidar y demostrar los conocimientos adquiridos en el máster.

2. INTRODUCCIÓN

2.1. Sistemas informáticos de apoyo al riego

2.1.1. *Estado* del arte

A nivel mundial, pero sobre todo en el ámbito mediterráneo, la eficiente gestión del riego es un aspecto estratégico para compatibilizar la sostenibilidad económica y medioambiental de la producción agrícola. Por ello es necesaria la optimización de las aplicaciones de riego a nivel de parcela, lo que requiere aportar las dosis justas en los momentos oportunos.

Pero conocer con precisión cuáles deben ser los aportes para cada sector de riego en un momento dado no es una cuestión sencilla. Las necesidades hídricas de los cultivos varían a lo largo de su desarrollo dependiendo de la fase fenológica, las condiciones meteorológicas y otros condicionantes específicos de cada escenario. La aplicación de estrategias de riego eficiente suele estar limitada por la formación, la dedicación y los medios que requeriría por parte de los regantes. Ellos plantean la necesidad de herramientas que ayuden a gestionar el riego, poniendo en juego todo el conocimiento e información disponibles para la toma de decisiones. En los últimos años, los avances en sensorización, comunicaciones y procesamiento de datos han creado la oportunidad de abordar la programación automatizada del riego desde “la nube”, usando plataformas web con capacidad de procesar cualquier tipo y volumen de datos, y accesibles desde cualquier dispositivo conectable a Internet.

Actualmente, existe un abanico de posibles tipologías de soporte al regante, según el nivel de complejidad y el coste económico que estarían dispuesto a asumir. Estas tipologías se podrían resumir en las siguientes:

Recomendaciones de agencias públicas. Proporcionan información generalista, gratuita y unidireccional. Se supone que son recomendaciones contrastadas y comercialmente imparciales. Ejemplos:

- Sistema de Información Agroclimática para el Regadío del Ministerio de Agricultura y Pesca, Alimentación y Medio Ambiente (MAPAMA) (http://eportal.mapama.gob.es/websiar/Inicio.aspx);

- Eina de reg de RuralCat (http://ruralcat.gencat.cat/web/guest/eines/eina-recomanacions-de-reg-agricultura)
- Sistema de Atención al Regante, Junta de Andalucía (www.juntadeandalucia.es/agriculturaypesca/ifapa/servifapa);
- InfoRiego, Junta de Castilla y León (www.inforiego.org);
- Red de Asesoramiento al Regante de la Junta de Extremadura, REDAREX (http://redarexplus.gobex.es/RedarexPlus/);

Internet 2.0, especialmente blogs especializados, foros y redes sociales. Proporcionan información de manera dinámica e interactiva y son una vía efectiva y sin coste de informar de eventos y novedades. Idealmente podrían encaminar al regante hacia FAQs, manuales, experiencias de otros regantes, etc., pero también pueden ofrecer un exceso de información, así como información no contrastada u orientadas de manera tendenciosa a determinados productos comerciales. Ejemplos:

- Homo agrícola (http://elhocino-adra.blogspot.com.es/);
- Agricultures. Red de especialistas en agricultura (http://agriculturers.com/);

Aplicaciones gratuitas o de bajo coste para móvil u ordenador. Pueden proporcionar, hasta cierto punto, recomendaciones de riego/fertirriego personalizadas a la casuística de cada explotación. El regante proporciona los datos de entrada específicos de cada finca (características del cultivo, suelo, sistema de riego, etc.) y la meteorología la aplicación suele obtenerla directamente de la red. Su fiabilidad dependerá de la fiabilidad de los datos entrados por el usuario y de que la casuística de interés esté bien soportada por la aplicación. Ejemplos:

- SIAR (Sistema de Información Agroclimática para el Regadío) (http://eportal.mapama.gob.es/websiar/Inicio.aspx);
- Aquadaia (www.aquadaia.com);

- RiegoApp (https://play.google.com/store/apps/details?id=com.iriego.riegoapp);

Servicios de asesoramiento agronómico, con técnicos que visiten la explotación. Permiten un mayor grado de personalizar las prácticas de riego/fertirriego a la casuística de cada parcela, con el respaldo de un profesional que le confiere fiabilidad. Puede apoyarse en un amplio repertorio de tecnologías (analíticas, sensores, teledetección, etc.) y presentan un coste que dependerá de las tecnologías usadas y de la frecuencia de las visitas a campo. Ejemplos:

- Saf Sampling (http://safsampling.com/);
- AKIS (http://akisinternational.com/es/);

Plataformas de supervisión y control ligadas al equipamiento. Incluyen la instalación de sensores con o sin automatización de riego. Proporciona datos objetivos sobre los que basar el manejo personalizado del riego/fertirriego de cada parcela, así como la validación necesaria para poder reajustarlos a lo largo de la campaña, Pueden usarlas directamente el regante o estar enmarcadas dentro un servicio de asesoramiento. Estas plataformas permiten reducir el número de visitas de los técnicos a la finca, pero implican una inversión en equipamiento. El sistema más avanzado llegaría hasta la integración de la monitorización, la automatización de la toma de decisiones y a comunicación con los dispositivos de telecontrol, por ejemplo, EFFIDRIP (www.effidrip.eu). A nivel comercial este nivel más avanzado de integración aún está muy limitado. Ejemplos:

- Bynse (http://bynse.com/);
- ModPow (www.modpow.es);

2.1.2. *Agricultura* de precisión

Esta aplicación se enmarca en el ámbito de la agricultura de precisión aplicada al riego. La agricultura de precisión se define por la posibilidad de

aplicar el recurso adecuado (agua, fertilizantes, fitosanitarios…) en la cantidad adecuada, en el momento y lugar precisos para cada caso en particular. Por lo tanto, pone énfasis en el conocimiento y la descripción de cada casuística particular (cultivo, suelo, microclima, instalaciones y maquinaria de la finca, prácticas agronómicas…). Para controlar y supervisar las aplicaciones en el lugar y momento preciso se utilizan sensores, mapas y mediciones georreferenciadas. La instalación de sensores fijos suele proporcionar un seguimiento en continuo de una cierta variable en un punto muy concreto. Mientras que los mapas (pueden provenir de teledetección o de la recopilación de muchas mediciones georreferenciadas) suelen proporcionar información continua en el espacio, correspondiente a un momento concreto.

2.1.3. Aplicación web base

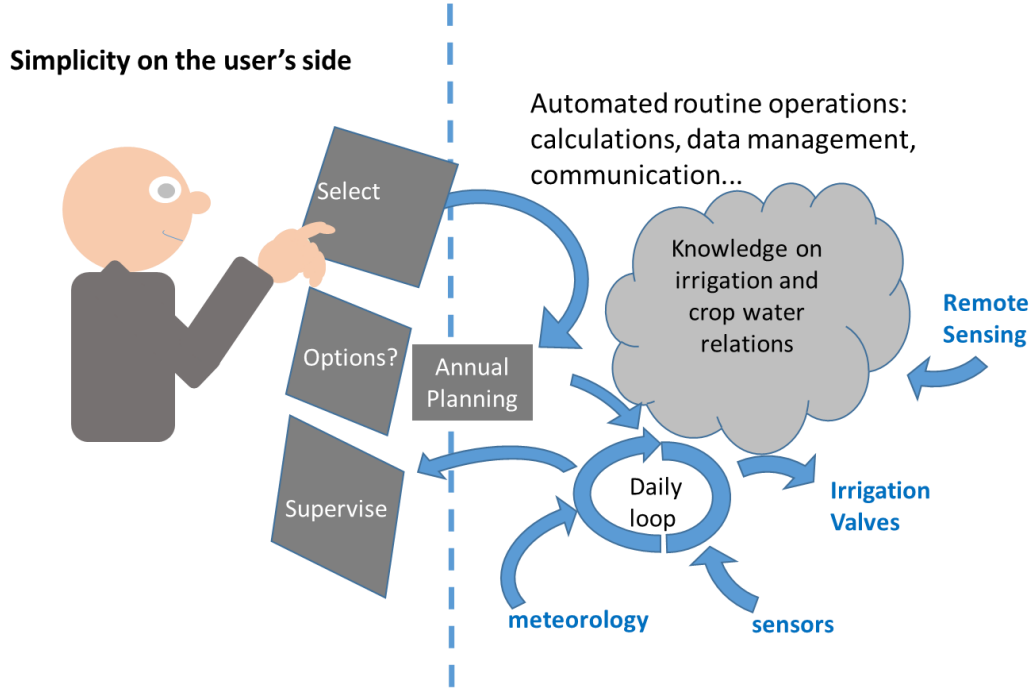
En el IRTA, dentro del programa de Uso Eficiente de Agua en la Agricultura, llevan varios años investigando cómo deben ser las herramientas informáticas de apoyo al riego. Como resultados de esas actividades han desarrollado diversas aplicaciones no comerciales donde estudian la eficacia de diversas aproximaciones para integrar tecnologías basadas en sensores con simulación de cultivos y con teledetección.

Algunos ejemplos de las aplicaciones actuales son IRRIX y Simulareg.

2.1.4. IRRIX

Es una aplicación web para la supervisión y el control automáticos del riego. La funcionalidad de IRRIX está dirigida a ajustar automáticamente cada día las dosis de riego a las necesidades particulares de cada sector de riego, teniendo en cuenta el clima, la disponibilidad de agua del suelo y el estado del cultivo. IRRIX integra los conocimientos científicos y la dinámica del agua del suelo, junto con las funcionalidades del análisis de datos del sensor y la interacción entre todos los elementos implicados a través de Internet.

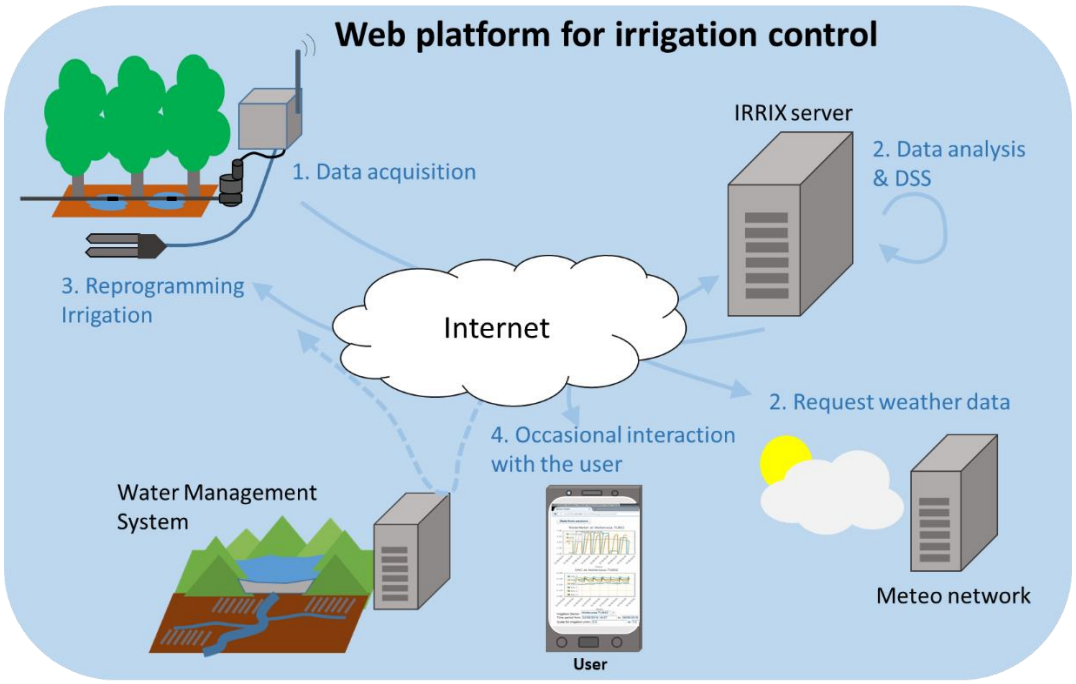
Figura 1 – IRRIX



Fuente: Proyecto IRTA

IRRIX está diseñado para administrar un conjunto grande y heterogéneo de escenarios de riego con el mínimo esfuerzo de dedicación por parte del productor. El flujo de trabajo anual consiste en concentrar la mayor parte de la participación del usuario fuera de temporada, en la preparación de un plan estacional que el sistema seguirá de forma autónoma al tiempo que lidia con las condiciones climáticas diarias y la retroalimentación de los sensores. Durante la temporada, el sistema interactúa a través de Internet con los dispositivos de adquisición de datos y la configuración de riego, instalados en el campo.

Figura 2 – Arquitectura plataforma web de controle de riego



Fuente: Proyecto IRTA

La interacción del usuario es a través de una aplicación web, accesible desde cualquier dispositivo con un navegador web. El caso de uso principal consiste en que los agricultores o los técnicos de riego tengan que administrar el riego para un gran número (decenas) de sectores de riego independientes.

2.1.5. Simulareg

Simulareg es una herramienta de aprendizaje para agricultores, técnicos de riego, expertos y todos aquellos que quieran aumentar su conocimiento.

El objetivo del simulador es educativo y proporciona un entorno para comprobar el comportamiento de una parcela o cultivo ligado a un calendario de programaciones de riego. Es posible definir parámetros como la meteorología, el tipo de suelo y el sistema de riego para una parcela y permite configurar distintas estrategias de riego siguiendo un calendario.

Simulareg funciona con la combinación de un simulador de cultivo y una base de datos de información basada en casos prácticos de riego. La herramienta, disponible en varios idiomas (Catalán, Castellano, Inglés, Francés, Italiano y Portugués), es de acceso libre a través de Internet mediante

la dirección www.ies-sim.com y se puede usar desde cualquier navegador. IES permite establecer un escenario con unos pasos muy sencillos, a través de los cuales, se definen las propiedades del cultivo, con más de 60 cultivos disponibles; el clima, con la posibilidad de utilizar datos históricos del Servicio Meteorológico de Cataluña y otras zonas de España y del mundo; el tipo de suelo, donde se pueden definir freáticos y salinidad; y la tipología y propiedades del sistema de riego, entre ellas las limitaciones que pueda tener, como por ejemplo una dotación de riego prefijada por la comunidad de regantes. En este primer paso se ha definido la casuística de la parcela virtual a simular.

En un segundo paso, el usuario entra las programaciones de riego que cree más convenientes para esta parcela de acuerdo con las características que ha especificado y la meteorología que se puede encontrar. Por su parte, el sistema dispone de una base de información donde se detallan más de 700 ejemplos de buenas prácticas en riego que previamente se han documentado y validado.

Cuando el usuario ha acabado de especificar las características de una parcela, el sistema IES busca el ejemplo que más se asemeja al nuevo caso de estudio. Una vez encontrado, el ejemplo es adaptado al caso de estudio y el patrón de riego resultante es usado para generar las recomendaciones del Experto Virtual.

El experto Virtual utiliza el patrón de riego generado y el estado (simulado) del cultivo para determinar cada día simulado la cantidad de agua necesaria. El estado del cultivo es el resultado de simular un conjunto de variables como, por ejemplo, el movimiento del agua al suelo en 2 dimensiones, con el objetivo de ilustrar qué parte del riego aportado sirve para atender las necesidades del cultivo y qué parte se va más allá del alcance de las raíces y se pierde por drenaje. También se simulan las pérdidas de producción, que se dan cada vez que el cultivo se encuentra en condiciones desfavorables.

Todo ello va encarado a facilitar la explicación de cómo una buena programación del riego permite obtener una buena producción y de calidad con un uso eficiente del agua.

2.2. Limitantes actuales de las aplicaciones de soporte al riego

En diversos proyectos de investigación en los que participa el IRTA se han demostrado aplicaciones web que adecúan automáticamente el riego a la casuística de cada parcela, la meteorología del día y los datos registrados por sensores (p.ej. EFFIDRIP, IRRIX, Giroreg). Sin embargo, a nivel comercial, actualmente el mercado ofrece solamente aproximaciones parciales que, o bien muestran datos de sensores que debe interpretar el mismo regante, o bien recomiendan de manera generalista cómo regar la próxima semana. Pero raramente integran la interpretación automatizada de los sensores con un reajuste del riego y, aún menos, interaccionan directamente con el sistema de riego para que éste adecúe las dosis a los resultados de la monitorización. Algunos posibles limitantes técnicos que explicaría por qué raramente se cierra el bucle de control automatizado del riego.

2.2.1. Enorme variedad de casuísticas

La combinación de posibles cultivos, tipos de suelo, instalaciones de riego y otros condiciones ambientales es muy elevada. Además, los casos de interés aplicado para las herramientas de control de riego suelen ser aquellos donde hay complicaciones añadidas, como son el riego con aguas moderadamente salinas, cultivos en zonas marginales con suelos de mala calidad, parcelas con variabilidad espacial, etc. Por todo ello, es un problema agronómicamente complejo diseñar una solución que sea adaptable a un repertorio amplio de escenarios de cultivo.

2.2.2. Dificultad de monitorizar e interpretar automáticamente los datos de los sensores

Existen distintos tipos de sensores que se pueden instalar en el suelo o en las plantas para monitorizar su estado hídrico. Pero en la práctica, la mayoría de estos sistemas son caros de instalar y sobretodo de mantener, porque suelen requerir revisar y manipular físicamente el sensor con mucha

asiduidad. Por ejemplo, sensores de flujo de savia, de micro-contracciones del diámetro de tronco, tensiómetros, etc. Por otro lado, existen algunos tipos de sensores de humedad del suelo (llamados sensores volumétricos de tipo capacitivo) que prácticamente no requieren otro mantenimiento que asegurar su alimentación eléctrica y conexión a algún sistema de comunicaciones, por lo que resultan de gran interés para el control automático del riego. Sin embargo, estos sensores detectan el contenido de agua a su alrededor, en un volumen muy reducido de suelo, alrededor de 1 L, que en la práctica puede ser poco representativo de lo que ocurre en todo el conjunto del suelo. Hay que tener en cuenta que el suelo es un medio muy heterogéneo y sin agitación, y lo que ocurre en un punto concreto puede ser distinto a lo que ocurra a pocos centímetros de distancia. Por ello, a pesar de que estos sensores registran muchos datos y que permiten que intuitivamente los regantes pueden ver tendencias, es difícil establecer sistemáticamente unos procedimientos de interpretación que sean válidos para un amplio repertorio de casos. El desarrollo de algoritmos de interpretación de sensores de humedad de suelo es precisamente uno de los temas de investigación en el IRTA.

2.2.3. Falta de integración de las tecnologías existentes

La situación actual es que hay poca interoperabilidad entre los distintos componentes que interesan para este tipo de aplicaciones: sensores en campo, agencias meteorológicas, herramientas de apoyo, dispositivos de control del riego… La integración es costosa y difícil porque los distintos componentes han aparecido por separado, cada uno con sus propios frameworks y estructuras de datos. Recientemente, con el auge del Internet of Things, estos problemas de interoperabilidad (no solamente en el ámbito del riego sino en el de la sensorización y actuación, en general) se han puesto muy de manifiesto y precisamente la adopción de estándares y la interoperabilidad entre sistemas es uno de los objetivos de asociaciones como la Alliance of Internet of Things (AIOTI).

2.2.4. Inmadurez de las soluciones comerciales hasta el momento

Entrevistas a usuarios potenciales realizadas por el IRTA revelan que la gran mayoría se muestran desconfiados y escépticos respecto a la oferta de soluciones existentes porque no ven claro que les resuelva algún problema práctico real. Como se ha comentado, suelen ser soluciones parciales que requieren demasiada intervención del usuario para completar el proceso. Por ejemplo, solamente muestran gráficos o mapas de datos primarios que el usuario debe interpretar y decidir qué hacer con ellos y la mayoría de veces no sabe qué hacer. En otras ocasiones, los usuarios más predispuestos a la innovación se sienten frustrados por soluciones muy simplistas y cerradas, que no les permiten interactuar aplicando su propio conocimiento.

En general, las soluciones existentes tienen un encaje difícil en los procedimientos de trabajo habituales para los regantes y, o bien son excesivamente simplistas y entonces no resuelven ningún problema real, o bien son más elaboradas y entonces los regantes se sienten incapaces de usarlas. Para superar este problema, la aproximación que se sigue en el IRTA es que, por un lado, el sistema pueda operar de forma autónoma incluso con nula participación del regante. Y además, considera un flujo de trabajo con distintos perfiles profesionales: 1) el instalador, que sabrá cómo desplegar los componentes y cómo iniciar el sistema en cada finca; 2) el consultor agronómico, que sabrá cómo configurar el sistema para resolver problemas reales que pueden ser distintos para cada caso concreto; 3) el regante, que si no quiere más, solamente necesita entender un panel de control que le indicará cómo progresa la campaña de riego y le avisará de posibles anomalías en la ejecución del riego previsto.

En cuanto a los desarrollos existentes en el IRTA, mayoritariamente se trata de una colección variopinta de códigos, resultante de distintos proyectos de investigación, que resulta adecuada para el uso interno y para demostraciones. Sin embargo, para afrontar nuevos retos en cuanto a integración de funcionalidades, escalado a un mayor volumen de trabajo y posible uso por terceros es necesario refactorizar algunas aplicaciones existentes. Un aspecto relevante a potenciar es la integración de las distintas líneas de desarrollo que había hasta el momento. Por ello, de la situación

actual en que el código se reparte entre Java, Python y Matlab se desea pasar a usar mayoritariamente un solo lenguaje de programación y que expertos del dominio con poca experiencia en programación también puedan aportar sus fragmentos de código.

3. TECNOLOGÍAS DISPONIBLES PARA EL DESARROLLO

3.1. Lenguaje Python

Este apartado hace referencia a (archive, 2018), (Venners, 2018), (Foundation., 2018).

Administrado por *Python Software Foundation*, soporte de orientación respecto a objetos, programación imperativa y programación funcional, es un lenguaje interpretado, tipificado, dinámico y multiplataforma.

Uno de los lenguajes de programación dinámicos más populares que existen entre los que se encuentran Perl, Tcl, PHP y Ruby. Aunque es considerado a menudo como un lenguaje “scripting”, es realmente un lenguaje de propósito general.

Es un lenguaje multiparadigma, significa que permite establecer varios estilos: programación orientada a objetos, programación imperativa y programación funcional, otros paradigmas tienen soporte mediante uso de extensiones.

3.1.1. Modo interactivo

El intérprete de Python estándar incluye un modo interactivo en el cual se formulan las instrucciones a través de una especie de intérprete de comandos, pudiendo introducir expresiones una a una y verse el resultado de su evaluación inmediatamente, la Figura 3 demuestra el uso de modo interactivo.

Figura 3 – Modo interactivo Python

```
>>> 1 + 1
2
>>> a = range(10)
>>> print( list(a) )
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3.1.2. Elementos de lenguaje

El contenido de los bloques de código (bucles, funciones, clases, etc) es delimitado mediante espacios o tabuladores, conocidos como tabulación, antes de cada línea de órdenes pertenecientes al bloque, Python se diferencia así de otros lenguajes. Un ejemplo de comparación entre C y Python queda reflejado en la Figura 4, donde se implementa el cálculo del factorial.

Figura 4 – Comparación entre C y Python

```c
int factorial(int x)
{
    if (x < 0 || x % 1 != 0) {
        printf("x debe ser un numero entero mayor o igual a 0");
        return -1; //Error
    }
    if (x == 0) {
        return 1;
    }
    return x * factorial(x - 1);
}
```

```python
def factorial(x):
    assert x >= 0 and x % 1 == 0, "x debe ser un entero mayor o
igual a 0."
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

Los comentarios se pueden poner de dos formas. Para comentarios largos se utiliza la notación ´´´comentario´´´, tres apóstrofos de apertura y tres de cierre. La otra notación es el símbolo #, el cual se extiende hasta el final de la línea.

Las variables se definen de forma dinámica, o sea que no se tiene que especificar su tipo de antemano y puede tomar distintos valores.

Los tipos de datos pueden ser divididos en tres categorías. Los mutables (list, set, dict) cuyo contenido puede cambiarse en tiempo de ejecución. Los inmutables (str, tuple, frozenset) cuyo contenido no puede cambiarse en tiempo de ejecución. Los naturales (int, long, float, bool, complex).

Los condicionales se definen usando la palabra clave *if* seguida de la condición, y el bloque de código. Las condiciones adicionales se introducen usando *elif* y su bloque de código. La Figura 5 demuestra el uso en cascada de los condicionales.

Figura 5 – Ejemplo de condicionales

```
>>> verdadero = True
>>> if verdadero: # No es necesario poner "verdadero == True"
...     print "Verdadero"
... else:
...     print "Falso"
...
Verdadero
>>> lenguaje = "Python"
>>> if lenguaje == "C": # lenguaje no es "C", por lo que este bloque se obviará y evaluará la siguiente condición
...     print "Lenguaje de programación: C"
... elif lenguaje == "Python": # Se pueden añadir tantos bloques "elif" como se quiera
...     print "Lenguaje de programación: Python"
... else: # En caso de que ninguna de las anteriores condiciones fuera cierta, se ejecutaría este bloque
...     print "Lenguaje de programación: indefinido"
...
Lenguaje de programación: Python
>>> if verdadero and lenguaje == "Python": # Uso de "and" para comprobar que ambas condiciones son verdaderas
...     print "Verdadero y Lenguaje de programación: Python"
...
Verdadero y Lenguaje de programación: Python
```

El bucle *for* es similar a *foreach* en otros lenguajes. Se define con la palabra clave *for* seguida de un nombre de variable, seguido de *in*, seguido del iterable, y finalmente el bloque de código interno. La Figura 6 demuestra el uso del bucle *for*.

Figura 6 – Ejemplo bucle for

```
>>> lista = ["a", "b", "c"]
>>> for i in lista: # Iteramos sobre una lista, que es iterable
...     print i
...
a
b
c
>>> cadena = "abcdef"
>>> for i in cadena: # Iteramos sobre una cadena, que también es iterable
...     print i, # Añadiendo una coma al final hacemos que no introduzca un salto de línea, sino un espacio
...
a b c d e f
```

Las *listas* y las *tuplas* pueden contener elementos de diferentes tipos. No obstante, las *listas* suelen usarse para elementos del mismo tipo en cantidad variable mientras que las *tuplas* se reservan para elementos distintos en cantidad fija. La Figura 7 demuestra el uso de las *listas* y *tuplas*.

Figura 7 – Ejemplo de listas y tuplas

```
>>> lista = ["abc", 42, 3.1415]
>>> lista[0] # Acceder a un elemento por su índice
'abc'
>>> lista[-1] # Acceder a un elemento usando un índice negativo
3.1415
>>> lista.append(True) # Añadir un elemento al final de la lista
>>> lista
['abc', 42, 3.1415, True]
>>> del lista[3] # Borra un elemento de la lista usando un índice (en este caso: True)
>>> lista[0] = "xyz" # Re-asignar el valor del primer elemento de la lista
>>> lista[0:2] # Mostrar los elementos de la lista del índice "0" al "2" (sin incluir este último)
['xyz', 42]
>>> lista_anidada = [lista, [True, 42L]] # Es posible anidar listas
>>> lista_anidada
[['xyz', 42, 3.1415], [True, 42L]]
>>> lista_anidada[1][0] # Acceder a un elemento de una lista dentro de otra lista (del segundo elemento, mostrar el primer elemento)
True


>>> tupla = ("abc", 42, 3.1415)
>>> tupla[0] # Acceder a un elemento por su índice
'abc'
>>> del tupla[0] # No es posible borrar (ni añadir) un elemento en una tupla, lo que provocará una excepción
( Excepción )
>>> tupla[0] = "xyz" # Tampoco es posible re-asignar el valor de un elemento en una tupla, lo que también provocará una excepción
( Excepción )
>>> tupla[0:2] # Mostrar los elementos de la tupla del índice "0" al "2" (sin incluir este último)
('abc', 42)
>>> tupla_anidada = (tupla, (True, 3.1415)) # También es posible anidar tuplas
>>> 1, 2, 3, "abc" # Esto también es una tupla, aunque es recomendable ponerla entre paréntesis (recuerda que requiere, al menos, una coma)
(1, 2, 3, 'abc')
>>> (1) # Aunque entre paréntesis, esto no es una tupla, ya que no posee al menos una coma, por lo que únicamente aparecerá el valor
1
>>> (1,) # En cambio, en este otro caso, sí es una tupla
(1,)
>>> (1, 2) # Con más de un elemento no es necesaria la coma final
(1, 2)
>>> (1, 2,) # Aunque agregarla no modifica el resultado
(1, 2)
```

Las funciones se definen con la palabra clave *def*, seguida del nombre de la función y sus parámetros. La Figura 8 demuestra cómo crear una función.

Figura 8 – Ejemplo de funciones

```
>>> def suma(x, y = 2):
...     return x + y # Retornar la suma del valor de la variable "x" y el valor de "y"
...
>>> suma(4) # La variable "y" no se modifica, siendo su valor: 2
6
>>> suma(4, 10) # La variable "y" sí se modifica, siendo su nuevo valor: 10
14
```

Las clases se definen con la palabra clave *class*, seguida del nombre de la clase, y si hereda de otra clase, el nombre de esta. El método “__init__”” se ejecuta al instanciar la clase, al igual que todos los métodos en Python, debe tener al menos un parámetro, generalmente se utiliza el *self*. La Figura 9 demuestra cómo definir una clase y crear una instancia de la misma.

Figura 9 – Ejemplo de cómo definir clase

```
>>> class Persona(object):
...     def __init__(self, nombre, edad):
...         self.nombre = nombre # Un atributo cualquiera
...         self.edad = edad # Otro atributo cualquiera
...     def mostrar_edad(self): # Es necesario que, al menos, tenga un parámetro, generalmente: "self"
...         print self.edad # mostrando un atributo
...     def modificar_edad(self, edad): # Modificando Edad
...         if edad < 0 or edad > 150: # Se comprueba que la edad no sea menor de 0 (algo imposible), ni mayor de 150 (algo realmente difícil)
...             return False
...         else: # Si está en el rango 0-150, entonces se modifica la variable
...             self.edad = edad # Se modifica la edad
...
>>> p = Persona("Alicia", 20) # Instanciar la clase, como se puede ver, no se especifica el valor de "self"
>>> p.nombre # La variable "nombre" del objeto sí es accesible desde fuera
'Alicia'
>>> p.nombre = "Andrea" # Y por tanto, se puede cambiar su contenido
>>> p.nombre
'Andrea'
>>> p.mostrar_edad() # Se llama a un método de la clase
20
>>> p.modificar_edad(21) # Es posible cambiar la edad usando el método específico que hemos hecho para hacerlo de forma controlada
>>> p.mostrar_edad()
21
```

3.2. Frameworks Python

Existe muchas opciones de frameworks de aplicaciones web en Python, algunas de ellos son, Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, Falcon, etc.

Algunos de estos frameworks son consideras microframeworks, poseen estructuras minimalistas.

Los tres frameworks más utilizados basados en los números de proyectos en Github y preguntas en StackOverflow son Django, Flask y Tornado. La Figura 10 demuestra el uso de estos frameworks.

Figura 10 – Clasificación de los frameworks Python

Framework	Score
Django	92
Flask	84
Tornado	73
Bottle	65

Fuente: http://hotframeworks.com/languages/python

3.2.1. Django

Este apartado hace referencia a (Mozilla, 2018).

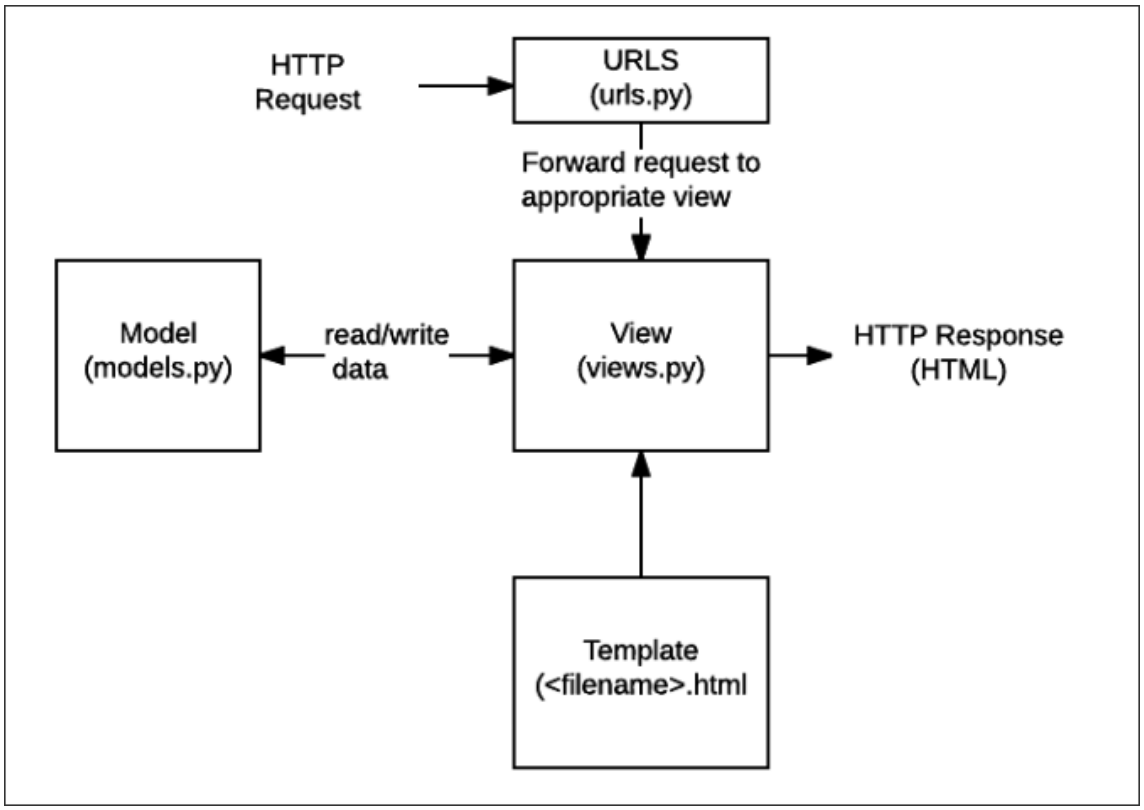
Es un framework de desarrollo web de código abierto diseñado con un patrón de proyecto modelo vista controlador con objetivo de facilitar la creación de sitios web. Hay algunas características de Django que te ayudan en la creación:

- Completo: Sigue la filosofía “baterías incluidas” es decir contiene estructuras y configuraciones básicas de cualquier sitio web, sigue principios de diseño consistentes y tiene una amplia y actualizada documentación.
- Versátil: Funciona con cualquier framework cliente-servidor, puede devolver contenido varios formatos entre ellos HTML, Json, XML, etc. Ya viene con algunos motores de base de datos como, Mysql, Oracle, SqlLite y etc. Viene con motores para plantillas.
- Seguro: Diseñado para proteger el sitio web automáticamente evitando así errores de seguridad. Incluye la inyección SQL, scripts entre sitios, falsificación de solicitud entre sitios y clickjacking. Las contraseñas de acceso de usuario son encriptadas con un hash de contraseñas unidireccional (*cryptographic hash function*).
- Escalable: Cada parte de la arquitectura es independiente de las otras facilitando la separación entre las diferentes partes que permite escalar para aumentar el tráfico al agregar hardware en cualquier nivel: servidores de cache, de base de datos o aplicación.
- Mantenible: Sigue el patrón de diseño MVC (modelo vista controlador). Esta escrito usando principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable.
- Portable: Está inscrito en Python, el cual se ejecuta en muchas plataformas, lo que significa que se puede ejecutar en varias distribuciones Linux, Windows y Mac OS X.

Django es “dogmático, pero no del todo”, es decir, proporciona un conjunto de componentes para gestionar la mayoría de las tareas de desarrollo y una o dos maneras preferidas de usarlos, sin embargo, si lo desea, la arquitectura desacoplada implica que puedes escoger y seleccionar de entre numerosas opciones diferentes o añadir soporte para otras completamente nuevas.

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en ficheros separados. La Figura 11 demuestra la arquitectura Django.

Figura 11 – Arquitectura MVT (*Model View Template*) Django



Fuente: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introducci%C3%B3n

- URLs: Se usa el mapeador URL para redirigir las peticiones HTTP a la vista apropiada, puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la vista como datos.
- Vista (*view*): Es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP, acceden a los

datos que necesitan y delegan el formateo de la respuesta a las plantillas (*templates*).

- Modelos (*models*): Son objetos Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar y consultar registros de la base de datos.
- Plantillas (*templates*): Es un fichero de texto que define la estructura o diagrama de otro fichero. Puede crear dinámicamente una página usando una plantilla, rellenándola con datos de un modelo.

Django se refiere a este tipo de organización como Modelo Vista Plantilla "Model View Template (MVT)" que es muy similar a la arquitectura MVC.

3.3. Base de datos

Este apartado hace referencia a (gestionbasesdatos readthedocs, 2018), (Herranz Gómes, 2014), (Telefonica, 2018), (Pandorafms, 2018), (Stonebraker, 2010).

Es una estructura organizada de datos que permite la extracción de informaciones, también formado por metadatos. Son operados por los SGBD (Sistemas Gestor de Base de Datos).

SGBD es un software que controla y gestiona una base de datos, facilita la utilización de la misma a muchos usuarios simultáneos y mantiene la integridad de los datos.

El conjunto de componentes o herramientas conceptuales que un SGBD proporciona para modelar recibe el nombre de modelo de BD. Los modelos más utilizados son el modelo relacional, el modelo jerárquico, el modelo en red, el modelo relacional con objetos y el modelo no relacional.

3.3.1. SQL (*Structured Query Language*)

Es un lenguaje de consulta de base de datos relacional. El alcance de SQL incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas y el control de acceso a los datos.

Las principales características de las bases de datos SQL son:

- Menor redundancia debido a que se crea una relación entre las tablas (*foreing keys*).
- Acceso a los datos más eficiente;
- Permite describir la información de la base de datos gracias a los metadatos.
- Integridad de los datos y dificultad en obtener incoherencias entre ellos
- Mayor seguridad en los datos.

Características negativas:

- El control y la administración requiere de un software y hardware potente.
- Personal cualificado debido a la dificultad de manejo;
- Implantación larga y difícil debido a los puntos anteriores;

Estos son algunos tipos de base de datos relacional: MySQL, SQLServer y Oracle.

3.3.2. NoSQL (*Not Only SQL*)

Las principales características de las bases de datos NoSQL son:

- Se ejecutan en máquinas con pocos recursos.
- Escalabilidad horizontal, se añade más nodos para mejorar el rendimiento.
- Utiliza arquitectura distribuida facilitando manejar grandes cantidades de datos.
- No utilizan SQL como lenguaje de consultas
- No utilizan estructuras fijas como tablas para el almacenamiento de los datos

Características negativas:

- La mayoría de ellas no admiten funciones de fiabilidad, que son soportadas por base de datos relacionales.
- Con el fin de apoyar las características de fiabilidad y coherencia, los desarrolladores deben implementar su propio código, lo que agrega más complejidad al sistema.
- Incompatibilidad con consultas SQL.

Hay cuatro tipos de base de datos NoSQL:

- Documento - Los datos se almacenan como documentos. Los documentos se pueden describir como datos en el formato de clave-valor, como por ejemplo el estándar JSON.
- Columnas - Los datos se almacenan en líneas privadas de tabla en el disco, pudiendo soportar varias filas y columnas. También permiten sub-columnas.
- Grafos - Los datos se almacenan en forma de grafos (vértices y aristas).
- Clave-valor - Es la que aguanta más carga de datos, pues el concepto de él es que un determinado valor sea accedido a través de una clave identificadora única.

Estos son algunos bancos de datos no relacional: Aerospike, Apache cassandra, Amazon dynamoDB, MongoDB, HBase.

3.4. Frameworks Front-End

Son frameworks que sirven como herramientas para auxiliar en la creación de diseños de sitios web.

3.4.1. Bootstrap

Este apartado hace referencia a (Twitter, 2018).

Un framework web de código abierto para diseño del front-end de sitios webs. Contiene plantillas de diseño con tipografía, formularios, botones,

cuadros, menú de navegación y otros elementos de diseño basados en HTML y CSS, así como extensiones Javascripts incorporadas.

A partir de la versión 2.0 bootstrap soporta diseños responsive, esto significa que el diseño se adapta dinámicamente de acuerdo con las características del dispositivo.

Para usar bootstrap en el desarrollo del sitio web, basta con incluir las referencias de BoostrapCDN, la Figura 12 es un ejemplo donde se incluye referencias para css y javascript.

Figura 12 – Como incluir las librerías de estilo y java script bootstrap

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/css/bootstrap.min.css"
integrity="sha384-Smlep5jCw/wG7hdkwQ/Z5nLIefveQRIY9nfy6xoR1uRYBtpZgI6339F5dgvm/e9B"
crossorigin="anonymous">



<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/js/bootstrap.min.js"
integrity="sha384-o+RDsa0aLu++PJvFqy8fFScvbHFLtbvScb8AjopnFD+iEQ7wo/CG0xlczd+2O/em"
crossorigin="anonymous"></script>
```

El bootstrap por defecto añade estilos y configuraciones para que sea posible la normalización de estilos entre navegadores, algunas de ellas son: requiere de HTML5, requiere el tag *meta* para que la adaptación dinámica del diseño funcione, además otras configuraciones son realizadas internamente. La Figura 13 demuestra un ejemplo de estructura de básica de HTML.

Figura 13 – Estructura HTML básica

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/css/bootstrap.min.css"
          integrity="sha384-Smlep5jCw/wG7hdkwQ/Z5nLIefveQRIY9nfy6xoR1uRYBtpZgI6339F5dgvm/e9B"
          crossorigin="anonymous">

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
            integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
            crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
            integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
            crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/js/bootstrap.min.js"
            integrity="sha384-o+RDsa0aLu++PJvFqy8fFScvbHFLtbvScb8AjopnFD+iEQ7wo/CG0xlczd+2O/em"
            crossorigin="anonymous"></script>
  </body>
</html>
```

En la web https://getbootstrap.com/docs/4.1/getting-started/introduction/ hay toda la documentación sobre los componentes y ejemplos de usos de los mismos.

4. ANÁLISIS DE REQUISITOS

Una aplicación real como la que se describe tiene un gran número de casos de uso, en los que pueden intervenir diversos actores (instalador, agrónomo, regante). Sin embargo, para limitar la complejidad de este proyecto y centrarnos en los objetivos de analizar la arquitectura de la aplicación, aquí solamente se tratan dos casos de uso representativos.

Algunos otros casos de uso no analizados aquí:

- Alta por el instalador de los dispositivos hardware con los que interactúa el sistema (sensores, válvulas, autómatas).
- Planificación por el agrónomo (o regantes avanzados) de la próxima campaña de riego, antes de que el sistema empiece a controlar automáticamente.
- Consultas ampliadas, por cualquiera de los actores, con mayor detalle sobre los datos registrados, el estado actual del sistema o las previsiones para próximas fechas.

4.1. Requisitos funcionales

Requisito	RF1. Entrar características de la parcela necesarias para planificar el riego
Prioridad	Esencial
Descripción	Para obtener la previsión de riego el agrónomo / regante avanzado deberá informar algunos datos como: Información del suelo (soil definition); Información del tipo cultivo (crop definition); Información del sistema de riego (irrigation definition); El escenario que quiere realizar el análisis (spatial view); Muchos de los campos serán rellenados por defecto por el sistema, donde estos datos están guardados en la base de datos.

Requisito	RF1.1 Entrar características del suelo (Soil Definition)

Prioridad	Esencial
Descripción	Los datos obligatorios para el suelo son: • Tipo (selección entre 5 tipos predefinidos) • Profundidad (valor numérico, m); Datos opcionales: • Posible presencia de agua subterránea (booleano); • Contenido de piedras (valor numérico, %); • Más información sobre el tipo, cantidad de agua y calidad del suelo.

Requisito	RF1.2 Entrar características del cultivo (Crop Definition)
Prioridad	Esencial
Descripción	Los datos obligatorios del cultivo son: • Tipo (selección entre los tipos disponibles); • Opciones de cultivo (selección) • Edad de la plantación (valor numérico, years); • Distancia entre 3 líneas (valor numérico, m); • Producción potencial (valor numérico, Kg/ha); • Fecha aproximada de floración ; (fecha) • Fecha aproximada de cosecha ; (fecha) • Altura, anchura y profundidad de los árboles en la plantación; (valores numéricos, m)

Requisito	RF1.3 Entrar dados (Irrigation Definition)
Prioridad	Esencial
Descripción	Los datos obligatorios son: • Objetivo del gestor de riego (selección de un desplegable) • Líneas de riego por línea de cultivos; (valor numérico, m)

	• Caudal de los goteros(valor numérico, l/h); • Uniformidad del riego(valor numérico, %); • Distancia entre goteros (valor numérico, m); Información sobre disponibilidad de agua. • Fecha de inicio y fin del abastecimiento; (fechas) • Hora para cada día de la semana (horario) • Volumen máximo anual de riego (valor numérico, m3/ha);

Requisito	RF1.4 Entrar dados (Spatial View)
Prioridad	Esencial
Descripción	El usuario deberá seleccionar en un mapa el área de interés, marcando el perímetro de toda la finca, así como el de cada uno de los sectores de riego y de las distintas zonas de manejo dentro de un sector. También los puntos donde ha instalado sensores.

Requisito	RF2 Resultado del riego
Prioridad	Esencial
Descripción	Enseñar gráficos del escenario seleccionado con información del riego realizado y previsto. Para cada uno de los gráficos se podrá mirar detalles del estado del riego y también informaciones de cuanto ha sido el riego (por encima o por debajo del previsto)

Requisito	RF2.1 Resultado del riego (Synoptic View)
Prioridad	Esencial
Descripción	Grid con informaciones de riego que ha sido realizado y riego previsto.

Requisito	RF2.2 Resultado del riego (State View)
Prioridad	Esencial

Descripción	Detalles de uno de los escenarios enseñado en el RF2.1. Gráficos con informaciones adicionales.

Requisito	RF2.3 Resultado del riego (Seasonal View)
Prioridad	Esencial
Descripción	Gráficos con la perspectiva anual del riego realizado (acumulado hasta la fecha), respecto a los volúmenes de riego máximo y mínimo previstos en la planificación anual..

4.2. Requisitos no funcionales

Requisito	RNF1.1 Autocomplete de campos
Prioridad	Importante
Descripción	Habrá datos que el usuario del sistema no sabrá y entonces estos datos deberán ser rellenados por defecto de acuerdo con el tipo de cultivo, tipo de sistema de riego o tipo de suelo.

Requisito	RNF2.1 Cálculo de los resultados
Prioridad	Esencial
Descripción	Para una mejor análisis y previsión del riego, deberá utilizar datos de sensores instalados en la finca como: estado del suelo, humedad del aire, cantidad de riego realizado, también datos de estaciones meteorológicas y de estaciones de la región de la finca.

Requisito	RNF2.2 Resultado del riego (Synoptic View)
Prioridad	Esencial
Descripción	Los gráficos deben ser muy visuales, de manera que en un vistazo superficial el usuario identifique si existe alguna incidencia o bien todo sigue el curso previsto. Por ejemplo, usar distintos colores para la interpretación del riego: Rojo: significa que hubo algún problema y el riego aplicado fue menor de lo previsto;

	Violeta: Se aplicó el volumen de riegoprevisto; Azul: Hubo más riego de lo que estaba previsto; Transparente: El riego aún no ha sido realizado;

Requisito	RNF3. Manejo de datos poco estructurados
Prioridad	Esencial
Descripción	La descripción de las fincas y de los sectores de riego a tratar es poco estructurada porque existen muchas casuísticas posibles, algunas de ellas impensables en el momento de diseñar una aplicación web, por lo que hay que prever que habrá continuamente cambios para acoger nuevas casuísticas. El abanico de posibilidades debe cubrir desde explotaciones muy tradicionales y poco tecnificadas hasta explotaciones modernas, muy profesionalizadas y a menudo dirigidas a optimizar la producción de algún producto hortofrutícola muy especializado. Por ello, para describir con precisión la orientación productiva de una finca hay que tener en cuenta que para un mismo tipo de cultivo puede haber muchos enfoques tecnológicos posibles, así como distintas orientaciones de mercado. Al mismo tiempo, existen distintos sistemas de riego (a manta, aspersión, goteros...), cada uno con unos atributos descriptivos completamente distintos y en algunos casos complementados con sistemas de fertirrigación también muy variados. El hecho de que estas instalaciones se hayan ido montando ad hoc hace difícil describir con un formulario muy estructurado las particularidades más relevantes de cada caso.;

Requisito	RNF4. Capacidad de gestionar series temporales de valores numéricos, largas y densas.
Prioridad	Esencial
Descripción	Esta necesidad surge sobretodo a partir del uso de sensores para medir en continuo (p.ej. una medición cada minuto)

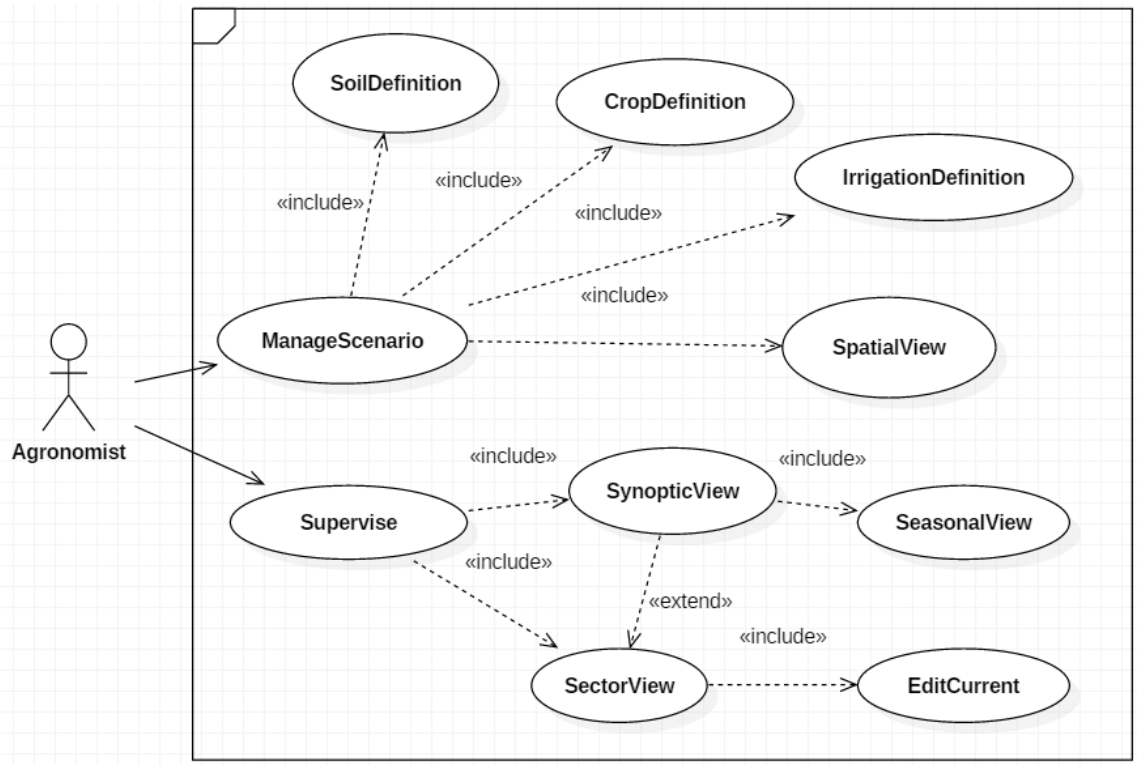
	algunas variables meteorológicas, del cultivo, del suelo o de la instalación de riego. Una finca tipo podría contener unas pocas variables meteorológicas (las más relevantes serían temperatura del aire, humedad relativa y pluviometría), un caudalímetro en cada sector de riego y varios sensores de humedad del suelo repartidos en distintos puntos de las parcelas. Generalmente estos datos se irán recopilando de manera incremental para que puedan ser analizados por períodos naturales (p.ej. una vez al día) para obtener la información relevante que alimentará un panel de control de los procesos que tienen lugar en la finca

Requisito	RNF5. Capacidad de gestionar datos geográficos.
Prioridad	Esencial
Descripción	Las actividades agrícolas tienen lugar sobre una extensión de terreno y es necesario saber en qué área hay que aplicar un determinado producto, o de qué punto proviene una determinada medición. De entrada, se considera que una finca está compuesta por varios sectores de riego, cada uno de los cuales se debe caracterizar en forma de polígono geográfico para conocer su ubicación y dimensiones. Además, si se instalan sensores habrá que conocer su ubicación exacta como puntos geográficos.

4.3. Descripción general de los casos de usos

El Agrónomo entrará la descripción de la finca y los detalles del escenario concreto para cada sector de riego, podrá acceder a los resultados del riego realizado y del que está por realizar, la previsión de riego anual y diversos detalles sobre estos indicadores. La Figura 14 demuestra la visión general de los casos de usos del actor *Agronomist*.

Figura 14 – Diagrama de caso de uso



4.4. Caso de Uso 1 Entrada descripción de finca

Este caso de uso hace referencia a *ManageScenario* de la Figura 14.

El Agrónomo entrará datos de la finca, estos datos están divididos en 4 grupos: definición del suelo, definición del sistema de riego, definición del cultivo y definición del contexto espacial.

Definir el tipo de suelo, hay datos que serán definidos por el sistema por si acaso el agrónomo no sepa más informaciones, si él quiere podrá modificarlos.

Definir el tipo de cultivo, hay datos que serán definidos por el sistema de acuerdo con el tipo de cultivo, el agrónomo también podrá modificarlos si quiere.

Definir el sistema de riego, definir fechas y horas, habrá datos que serán definidos por el sistema caso el agrónomo quiere podrá modificarlos.

Definir el contexto espacial donde se aplicará el riego, a partir de un mapa el agrónomo deberá seleccionarlo.

4.5. Caso de Uso 2 Salida de planificación anual de riego

Este caso de uso hace referencia a *Supervise* de la Figura 14.

Las salidas de los datos estarán detalladas por gráficos. Podrá tener una vista detallada (*SectorView*) de un determinado sector de riego y desde este sector ir a una vista general (*SynopticView*) de todos los sectores que maneja este usuario (en la misma finca o en otras). Desde la vista general (*SynopticView*) podrá acceder a la vista detallada (*SectorView*) de un sector de riego.

Desde la vista detallada (*SectorView*) podrá acceder a la vista de detalles de cómo evoluciona (*SeasonalView*) el riego acumulado hasta la fecha respecto a los máximos y mínimos previstos en la planificación anual.

5. DISEÑO DE SOLUCIÓN

5.1. Lenguaje de programación

La solución se centra en Python porque así lo indicó el IRTA. Python es actualmente el lenguaje de programación con mayor crecimiento. Específicamente, en entornos de investigación Python tiene un gran interés por la rápida curva de aprendizaje para profesionales de ámbitos muy diversos, por la disponibilidad de librerías especializadas para una amplia variedad de funcionalidades y en concreto por su adecuación a la exploración y análisis de grandes volúmenes de datos.

5.2. Diseño de la arquitectura

5.2.1. Back-End

Entre los frameworks Python fue elegido el framework Django. En comparación los otros, Django hay más soporte y mantenimiento, no es un microframework, o sea, que viene con una arquitectura más completa permitiendo así ahorrar tiempo con algunas capas del sitio web.

Para retornar via AJAX un modelo de datos se tiene que hacer por medio del framework Django Rest Framework, esto permite definir una clase que serializa los modelos de datos.

5.2.2. Front-End

El framework Django ya viene con soporte a plantillas así es muy sencillo heredar vista de otras vistas, también es una característica importante del Django las vistas basadas en plantillas, es conocido como MVT (*model, view, template*) modelo, vista, plantilla. Para el diseño ha sido aplicado el framework bootstrap que actualmente es uno de los principales framework de diseño, también HTML5 y CSS.

5.3. Diseño de la Base de datos

Como se ha comentado, la descripción de las fincas suele ser poco estructurada si se quiere recopilar cualquier tipo de información a tener en cuenta para el manejo del riego. Ello incluye detalles de su instalación de riego, información histórica y otros parámetros diversos que variarán de un caso a otro.

Otro aspecto a considerar es cómo almacenar en la base de datos las series temporales de datos registrados por los sensores. Hay que tener en cuenta que puede representar un elevado volumen de datos, siempre acumulativo y que la frecuencia de muestreo puede ser distinta de un sensor a otro.

Por la poca estructuración de la descripción de las fincas, así como la variedad de los datos que se obtiene de los sensores, por no haber un patrón y por la gran cantidad de datos que se puede obtener fue elegido la base de datos MongoDB, que es un tipo de base de datos NoSql, porque en estos tipos de base de datos hay una mejor flexibilidad para guardar datos ya que el crecimiento de los datos es en anchura.

Las series de datos obtenidos de los sensores serán almacenadas como un tipo JSON, empaquetando fragmentos de la serie, por ejemplo, para cada 1000 datos de lectura del sensor será registrado en base de datos el intervalo de fecha con el archivo JSON correspondiente a estas lecturas.

Figura 15 – Diagrama de la base de datos

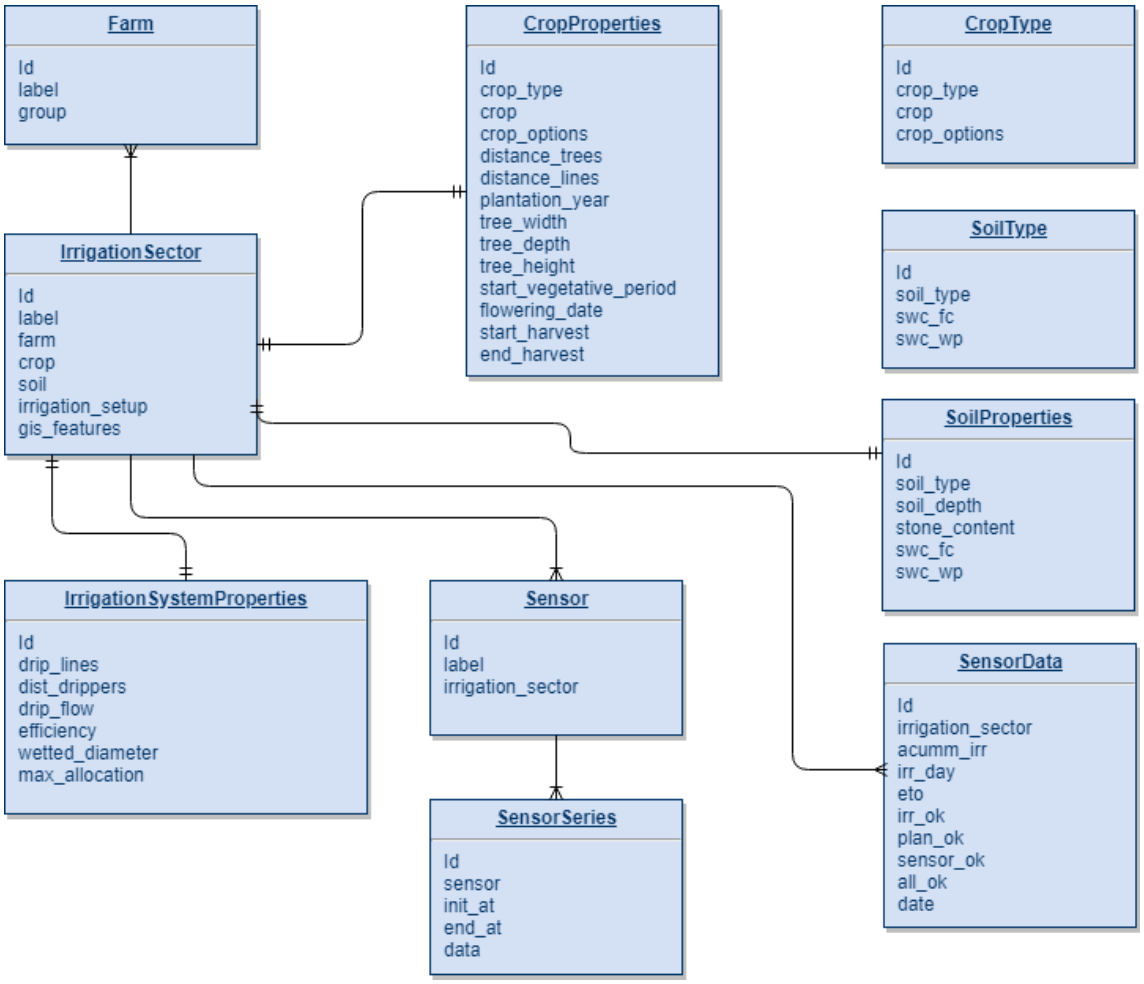


Tabla Farm, almacena datos de la finca y grupo de usuario que pueden acceder a informaciones sobre la finca.

Tabla IrrigationSector, almacena las parcelas o sector de una finca, tienes referencias a las tablas de sensores, cultivos, riego y suelo.

Tabla IrrigationSystemProperties, almacena datos del sistema de riego.

Tabla CropPropeties, almacena datos del cultivo.

Tabla SoilProperties, almacena datos sobre el suelo.

Tabla Sensor, almacena datos de los sensores que están en la parcela.

Tabla SensorSeries, almacena un archivo JSON en formato string con los datos del sensor, la fecha inicial y final indica el rango de los datos almacenados.

Tabla SensorData, almacena el resumen de los datos de los sensores, son indicadores para indicar estados del riego.

Tabla SoilType, almacena datos por defecto del tipo de suelo.

Tabla CropType, almacena datos por defecto de tipo de cultivo.

5.4. Soporte a series temporales de mediciones con sensores

Con el boom de IoT hay mucha necesidad de sistemas de BBDD que sean apropiados para las series de datos de sensores. Por ello, están en pleno auge algunos sistemas de bases de datos especializados y optimizados para series temporales de datos, Time Series Database (TSDB). Este tipo de base de datos es el que ha mostrado un mayor crecimiento en los últimos dos años (https://db-engines.com/en/ranking_categories). Algunas características de las TSDB son la capacidad de almacenamiento y compresión de datos con sello de tiempo, administración del ciclo de vida de los datos, resumen de datos, capacidad de manejar grandes escaneos dependientes de series temporales de muchos registros y consultas basadas en diferentes operaciones con la referencia de tiempo del dato. Algunos ejemplos de TSDB: Influx (www.influxdata.com), RDDtool (https://oss.oetiker.ch/rrdtool/), Graphite (https://graphiteapp.org/). Entre estos cabe destacar que Graphite resulta muy apropiado para uso con Python y se ofrece integrado con Django.

Otra aproximación es implementar las funcionalidades de tratamiento de datos temporales sobre cualquier otra base de datos, no TSDB, cortando la serie en trozos que se guardan como blobs. Por ejemplo, TimescaleDB es una extensión de PostgreSQL que implementa estas funcionalidades manteniendo también las funcionalidades de una base de datos relacional https://blog.timescale.com/. Una implementación parecida, sobre MySQL, es la que se usa en la aplicación IRRIX, del IRTA. De manera similar, se puede optimizar MongoDB para manejar series de datos temporales con Artic https://github.com/manahl/arctic. Más o menos, lo que hace Arctic es gestionar las timeSeries en trozos que guarda en MongoDB.

5.5. Soporte a datos geográficos

Por un lado, es necesario fijar el formato de los datos geográficos. El formato más extendido para la representación vectorial de datos espaciales es Shapefile, que es propiedad de ESRI. Por ello han aparecido recientemente

formatos más abiertos, que están teniendo bastante éxito. Uno de ellos es GeoJSON (http://geojson.org/), que sería un JSON mejorado y especializado para intercambiar datos GIS. GeoJSON define la gramática basada en un estándar del OGC (WKT), en el que expresamos tanto la geometría como los atributos asociados a dicha información.

Por otro lado, es necesario disponer de librerías que permitan interactuar con estos datos. Google ofrece librerías para implementar las interfaces de usuario en diferentes lenguajes de programación, en concreto para Python ofrece gmplot (https://github.com/vgm64/gmplot). Esta librería es fácil de usar pero para algunas aplicaciones puede ser demasiado simple y le faltarían algunas funcionalidades. Por otro lado, folium (https://github.com/python-visualization/folium) es una de las librerías más usadas en Python para mostrar datos GIS. Folium es muy fácil de usar para mostrar datos de salida. Sin embargo, para la entrada y edición gráfica de datos geográficos, realmente quien hace el trabajo es el javascript leaflet (https://leafletjs.com).

Otra opción a tener en cuenta es GeoDjango (https://docs.djangoproject.com/en/2.1/ref/contrib/gis/tutorial). GeoDjango es un módulo para Django que lo convierte en un potente framework para aplicaciones web con prestaciones de entrada y salida de datos geográficos. Una limitación práctica de GeoDjango para este proyecto es que está muy ligada a ciertas bases de datos (PostGIS, MySQL,) entre las que no se encuentra MongoDB.

Por su lado, geoPandas (http://geopandas.org) es una librería de Python que permite trabajar con datos geoespaciales de manera fácil. GeoPandas combina las funcionalidades de pandas (operaciones geoespaciales) con las de shapely (interfaz de alto nivel para geometrías múltiples). De esta manera, geoPandas ofrece para Python una alternativa para las funcionalidades de un Sistema de Información Geográfica.

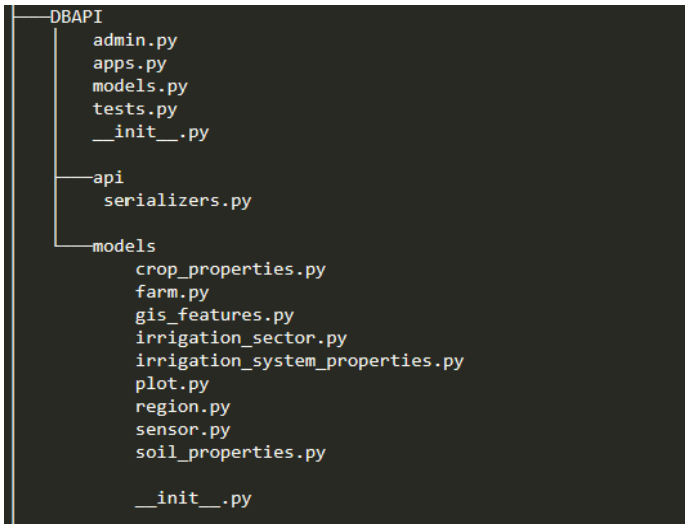
6. IMPLEMENTACIÓN

6.1. Arquitectura

La Figura 16 demuestra la estructura organizacional del API de acceso a la base de datos. Es un API para obtener los datos sin que la aplicación mezcle sus estructuras de datos con la estructura de la base datos.

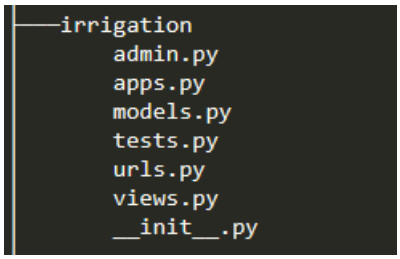
Las clases que definen los modelos que pueden ser retornados vía JSON están definidas en el archivo serializers.py en la carpeta api.

Figura 16 – Estructura organizacional del API de la base de datos



En Django se puede crear *apps*, apps son módulos dentro del proyecto principal, la estructura organizacional del módulo de riego está en la Figura 17.

Figura 17 – Estructura organizacional de la app irrigation

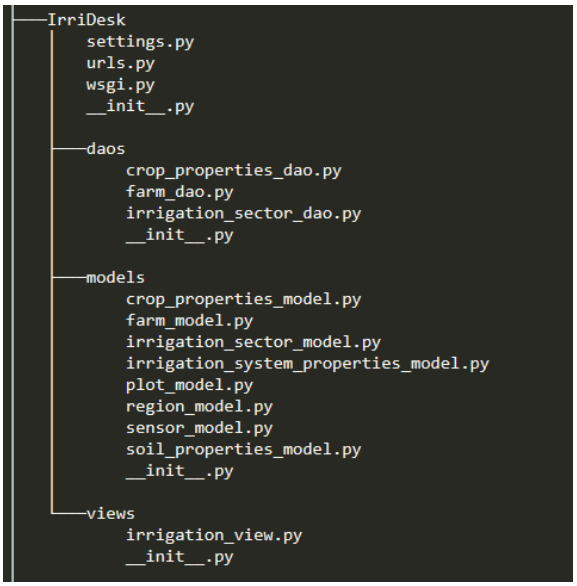


La Figura 18 demuestra la estructura organizacional del módulo principal del proyecto. Los controladores están definidos en carpeta view, para los casos de usos estudiados solo hace falta un controlador que es el irrigation_view.

Los modelos definidos en este módulo son modelos de la app y no de la base de datos, estos modelos están en la carpeta models.

La capa responsable de traspasar los datos de la base de datos para la aplicación es la dao, la clase responsable se encuentran en carpeta daos.

Figura 18 – Estructura principal del proyecto



La Figura 19 demuestra el diagrama UML de los modelos de la aplicación.

El modelo IrrigationSectorModel representa el sector o parcela de una finca, posee un objeto GeoJSON que son los datos geoespaciales de la parcela, una lista de sensores asignados a la parcela, una lista de los indicadores de los sensores, estos indicadores sirven para saber si hubo algún problema en la planificación de riego y otros indicadores, además guarda informaciones del riego, del cultivo y del suelo.

El modelo FarmModel representa una finca, posee grupo de acceso para los usuarios, esto grupos indicará las acciones que el usuario podrá realizar en la finca.

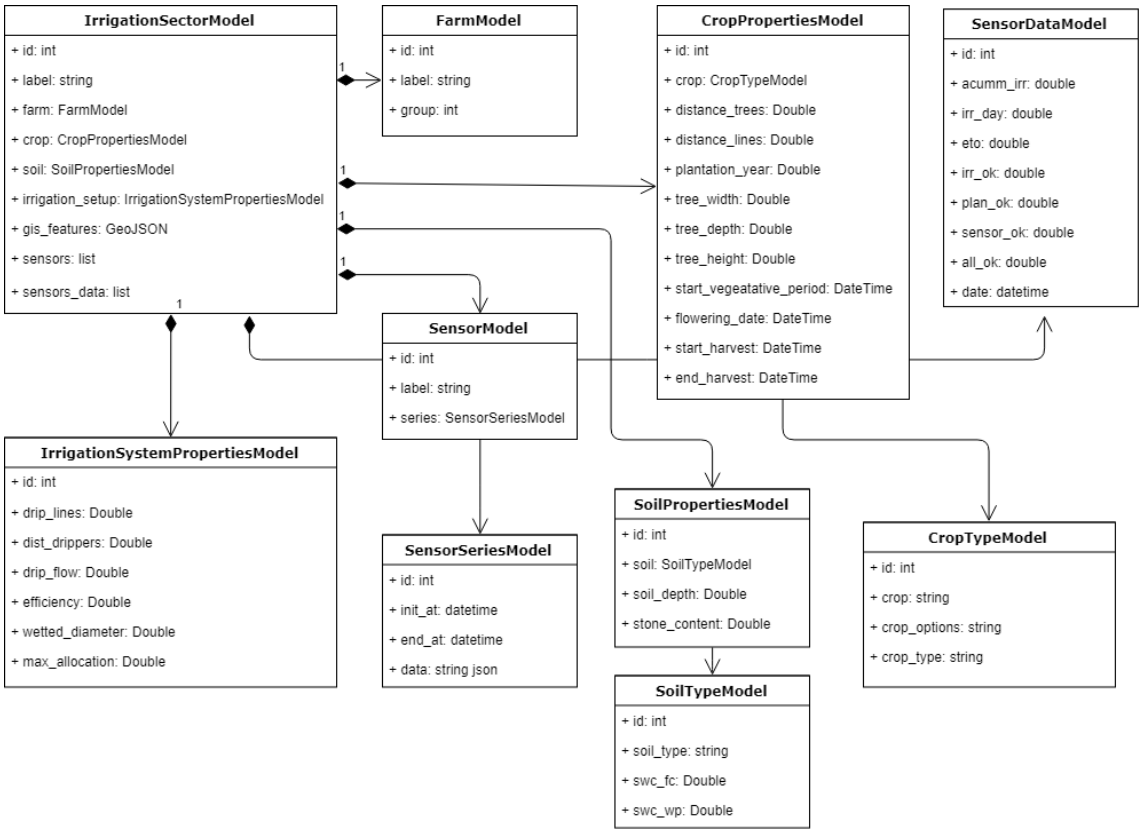
El modelo SensorModel representa los sensores de la parcela con una lista de series de datos agrupados por rango de fecha.

El model SensorSeriesModel representa una serie de datos del sensor con el intervalo de fecha de los datos que fueron agrupados en formato json, por ejemplo, puede ser un json con 1000 datos del sensor para un rango de fechas.

El model SensorDataModel representa un resumen de los datos leído de los sensores en una fecha. Son indicadores de estado de los sensores.

Los demás modelos son configuraciones del riego, informaciones del suelo y del cultivo y valores de cultivo y suelo que son datos al usuario por defecto.

Figura 19 – Diagrama UML de los modelos de la aplicación



6.2. Base de datos

El equipo de MongoDB dispone un engine que te permite conectar aplicaciones desarrolladas con Django a la base de datos mongo. Básicamente este engine hace una traducción de las querys sql para el lenguaje de MongoDB.

Django viene integrado con herramientas que auxilian en las comunicaciones con la base de datos para crear, editar y buscar datos sin que sea necesario escribir ninguna query. La Figura 20 muestra las tablas de la base de datos creadas desde la aplicación Django, y como el IDE del mongoDB las interpreta.

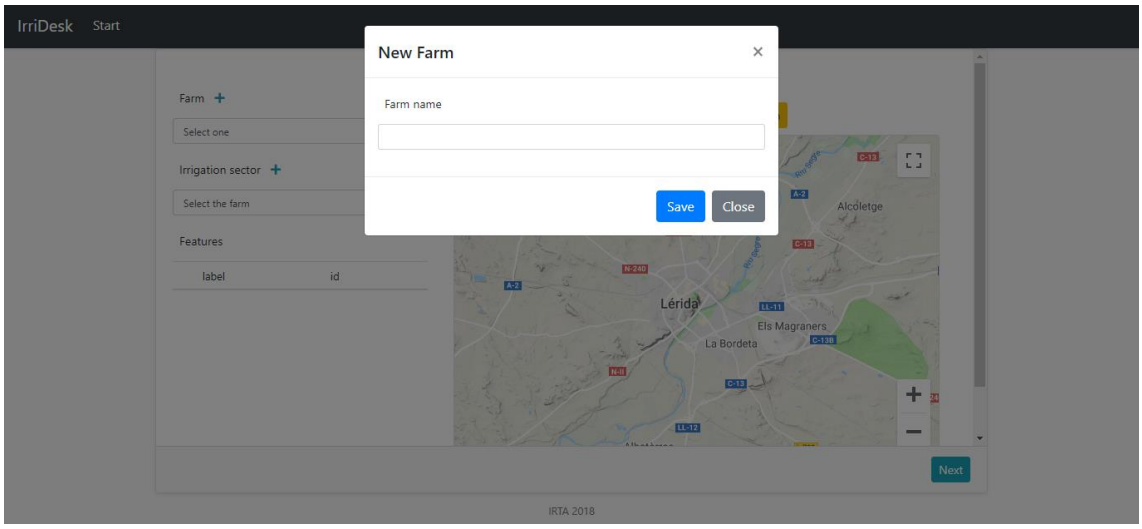
Figura 20 – Tablas de la base de datos

Collection Name ▲	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	
DBAPI_cropproperties	0	-	0.0 B	2	8.0 KB	
DBAPI_croptype	0	-	0.0 B	2	8.0 KB	
DBAPI_farm	0	-	0.0 B	3	12.0 KB	
DBAPI_irrigationsector	0	-	0.0 B	6	24.0 KB	
DBAPI_irrigationsystemproperties	0	-	0.0 B	2	8.0 KB	
DBAPI_sensor	0	-	0.0 B	2	8.0 KB	
DBAPI_soilproperties	0	-	0.0 B	2	8.0 KB	
DBAPI_soiltype	0	-	0.0 B	2	8.0 KB	
__schema__	17	98.3 B	1.6 KB	3	48.0 KB	

7. RESULTADOS, ANÁLISIS

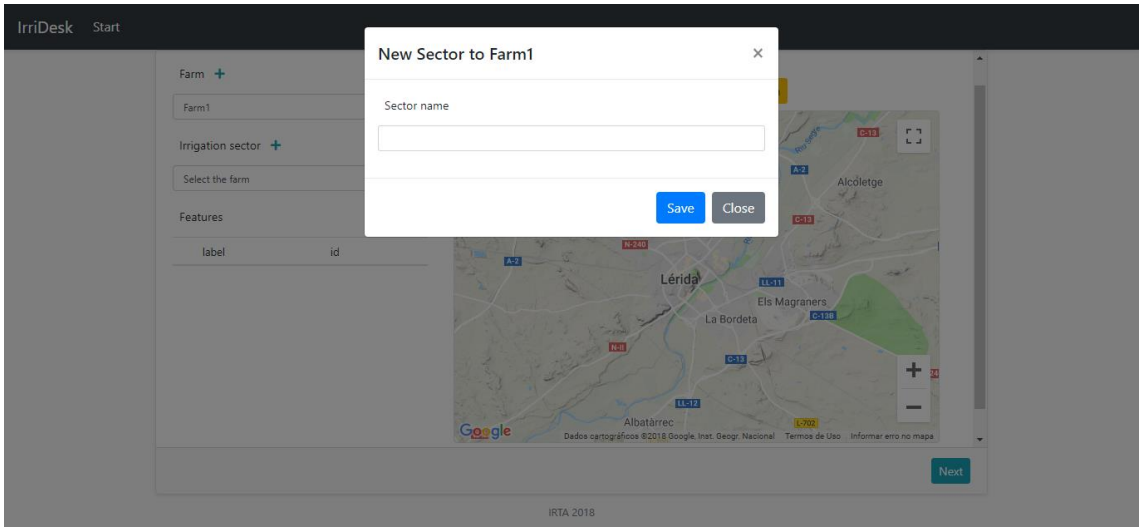
Para el caso de uso 1, el usuario debe registrar datos sobre la finca. La Figura 21, es la vista donde el usuario añade una nueva finca.

Figura 21 – Pantalla para crear finca



El usuario puede añadir parcelas a la finca seleccionada anteriormente, como muestra la Figura 22.

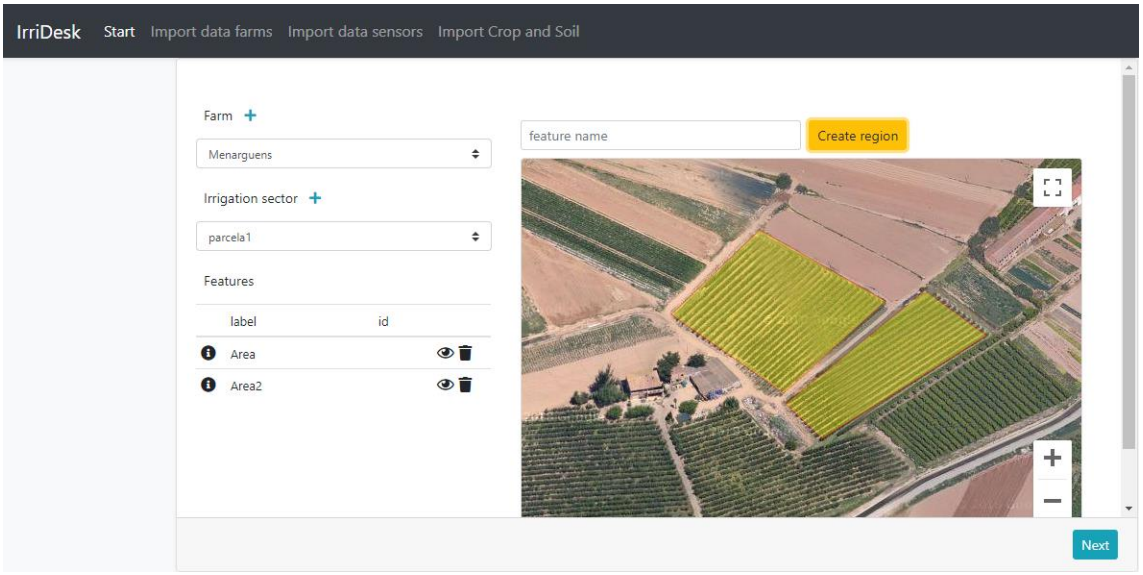
Figura 22 – Pantalla para crear parcelas



Después que se haya creado la finca y las parcelas, el usuario puede añadir datos geográficos de la finca y de las parcelas, donde cada parcela

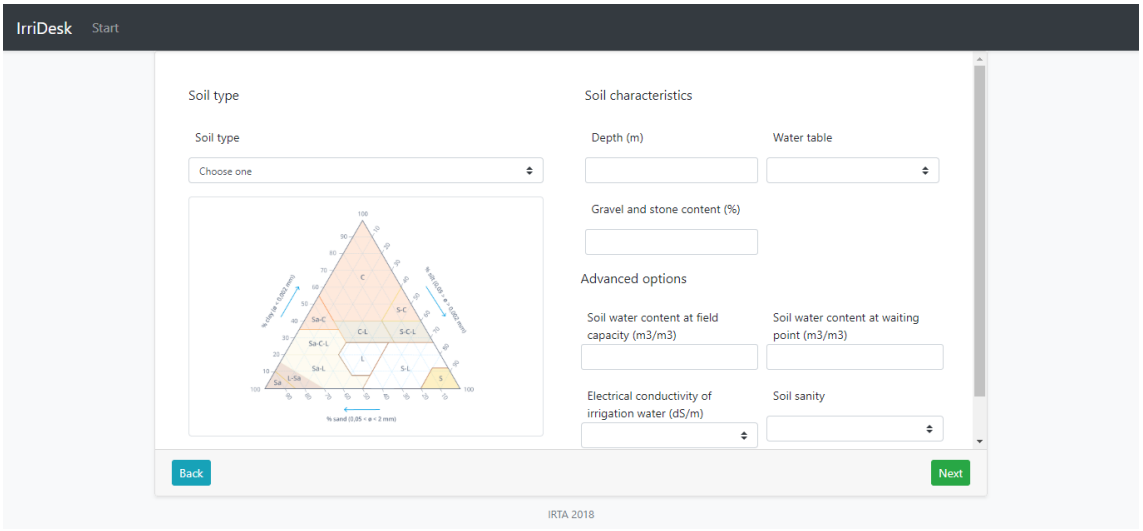
puede tener puntos y polígonos donde respectivamente representan sensores y finca o sub parcelas de la parcela, como demostrado en la Figura 23.

Figura 23 – Pantalla para definir datos geográficos de las parcelas



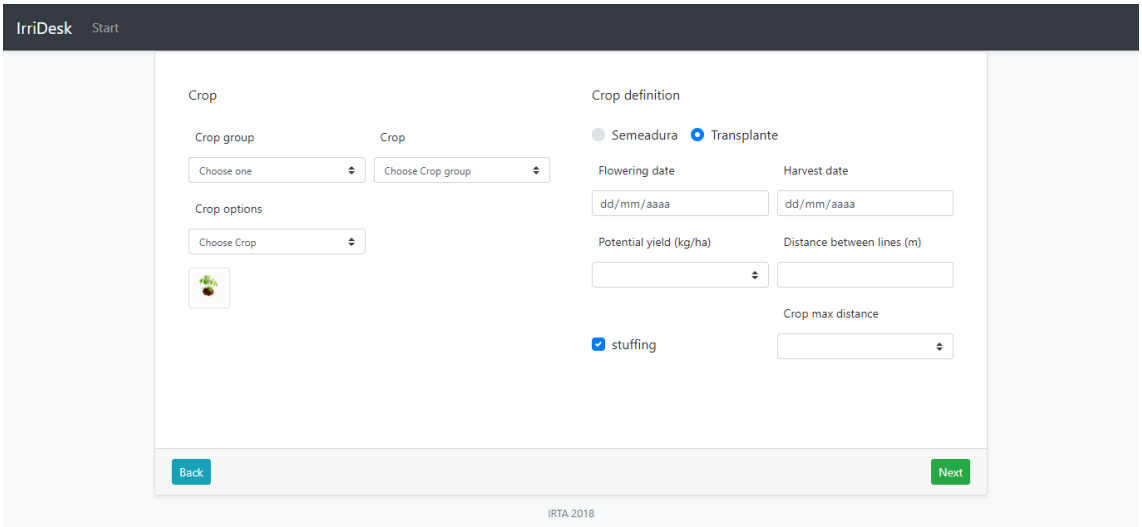
Una vez definidos los datos espaciales, se definen los datos del tipo de suelo, donde algunos datos ya viene rellenados por defecto, como muestra la Figura 24.

Figura 24 – Pantalla para definir tipo de suelo



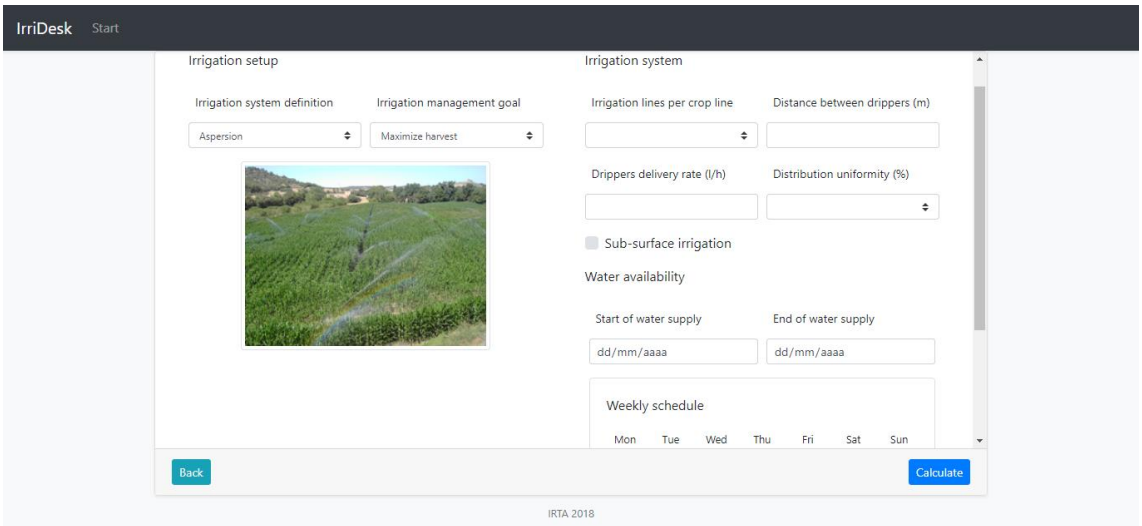
Tras haber definido el tipo de suelo, se define el tipo de cultivo, los tipos de cultivos y sus grupos están pre definidos, como muestra la Figura 25.

Figura 25 – Pantalla para definir tipo de cultivo



Después de definir el tipo de cultivo se define el tipo del sistema de riego, en este caso, solo es posible activar el tipo aspersión como demostrado en la Figura 26.

Figura 26 – Pantalla para definir tipo de riego



Una vez que haya acabado la entrada de los datos se muestra los resultados con los gráficos de los sensores y de los evapotranspiration (ETo). Se puede actualizar las referencias de ETo y se actualizan los datos en tiempo real. La figura siguiente muestra el resultado. Los gráficos presentados son

renderizados con ChartJS, es una librería open source para enseñar gráficos con javascript.

Figura 27 – Pantalla de resultado del riego

La Figura 28 muestra como MongoDB almacena los datos como estructura de archivo JSON.

Figura 28 – Ejemplo de cómo queda guardado un sector de riego en la base datos, a la izquierda. A la derecha, cómo queda guardado el contorno del sector.


```
{
  "_id":"5b82eee6310e4739a427d4e1",
  "id":1,
  "label":"Sector1",
  "farm_id":1,
  "crop_id":null,
  "soil_id":null,
  "irrigation_setup_id":null,
  "gis_features":{
    "type":"FeatureCollection",
    "features":[
      {
        "type":"Farm1",
        "properties":{

        },
        "geometry":{
          "type":"Polygon",
          "coordinate":[ ]
        }
      }
    ]
  }
}
```

```
"coordinate":[
  [
    41.524266710865305,
    0.545944796661388
  ],
  [
    41.523431334680986,
    0.5573602782287708
  ],
  [
    41.52060382757114,
    0.5549999342956653
  ],
  [
    41.52002545862426,
    0.5447860823669544
  ],
  [
    41.523110033276865,
    0.5433269606628528
  ]
]
```

El parámetro gis_features es un tipo string con el formato GeoJSON, dentro de este objeto puede haber una lista de tipos de coordenadas (Point, Polygon).

Para la definición del contexto espacial, las entradas de las coordenadas de las fincas con sus sectores y sensores han utilizado la librería de google Maps. Se captura las coordenadas GPS que son seleccionadas en el mapa formando un polígono.

Fueron encontradas otras librerías como:

- GMPlot y Folium, están bien, aunque, solo muestren datos GIS, pero sin posibilidad de editarlos;
- GeoDjango, es el recomendable en documentación oficial pero no resulta compatible con MongoDB;

Leaflet, es una librería javascript que te permite editar los datos, la cual utiliza también las librerías de google maps.

Las coordenadas capturadas son almacenadas en la base de datos en el formato GeoJSON, que es un estándar para tipos de datos GIS.

Para la definición de tipo de suelo, tipo de cultivo y sistema de riego, algunos datos serán rellenados por defecto basado en elecciones anteriores.

8. CONCLUSIÓN

La solución propuesta ha satisfecho adecuadamente los requisitos que se habían planteado. Concretamente, se ha implementado y testeado los requisitos funcionales correspondientes a los casos de uso en los que se ha centrado el trabajo. Además, se han resuelto y testeado satisfactoriamente algunos requisitos no funcionales que pueden ser habituales en aplicaciones para agricultura de precisión: manejo de datos no estructurados, soporte a series temporales de datos registrados por sensores, así como soporte a datos geográficos.

Hoy con el internet de las cosas (IoT), las series de datos es un problema muy común, es muy fácil superar las gigas de datos, debido a que no hay un estándar de los datos de los sensores, sus desarrolladores tienen sus propios frameworks, sus propios estándares de datos, por lo tanto, hay complejidad en almacenar los datos en la base de datos. MongoDB demuestra ser bastante flexible cuando no se tiene un patrón de datos o cuando la cantidad de los mismos crece radicalmente, sin embargo, MongoDB es una herramienta de pago para ser utilizada a nivel comercial y también puede haber algunas dificultades para integrarlo con Django, debido a que Django depende de un engine creado por MongoDB para que haya compatibilidad entre ellos donde la versión de Python también puede dificultar esta integración. Hay otras herramientas gratuitas para base de datos NoSQL y también hay un nuevo concepto de base de datos: newSQL que se podría estudiar para sustituirlo en la aplicación.

Python es un lenguaje muy utilizado actualmente debido a que es fácil integrarlo con otras aplicaciones y que posee métodos nativos para iterar listas y diccionarios que en otros lenguajes se tendría que definir clases o métodos. Django facilita el desarrollo web utilizando Python, su comunidad viene creciendo a cada día y es una de los frameworks para desarrollo web más utilizadas actualmente. Viene con un módulo admin que permite administrar los datos de la base de datos, sin tener la necesidad de implementar esta funcionalidad, con solo indicar el model que es responsable por tabla ya es posible añadir, editar o borrar datos de la misma. Posee muchas librerías indicadas en la documentación oficial en su web, sin embargo, no viene por

defecto con una librería para funcionar como un REST, que es un poco raro ya que hoy, es muy común retornar datos JSON por request AJAX. La aplicación recibirá contribuciones de terceros que están siendo desarrolladas con Python por lo tanto no habrá dificultad en esta integración. Lo que haría falta sería implementar los accesos al servicios externos bien como las llamadas a los módulos de estos terceros.

Este proyecto, a pesar de haber analizado solamente 2 casos de usos simples, demostró haber resuelto el nivel de complejidad planteado, principalmente en la arquitectura de la base de datos. Con los conocimientos adquiridos en este máster, seguramente lo mejor es que la arquitectura de la base de datos sea otro proyecto desacoplado de la arquitectura de la aplicación. Una solución sería implementar un modelo de base datos de acuerdo con los objetos que dispone MongoDB, la base de datos propuesta en este proyecto sigue el patrón de una base de datos relacional para que en el futuro puedan cambiar el motor de base de datos. Los datos de salida para el resultado de los gráficos fueron simplificados debido a que son muchos y tienen sus particularidades.

Para la continuidad del proyecto se tendría que estudiar más a fondo la arquitectura de la base de datos, si realmente el desarrollo seguirá con mongoDB entonces remodelar algunas tablas, modelos y daos. Para las entradas de los datos geográficos, del suelo, del cultivo y del riego, implementar las excepciones y particularidad de cada uno. Para la integración con terceros verificar si será a través de REST API o algún otro método.

9. REFERENCIAS BIBLIOGRÁFICAS

A. R. (20 de 07 de 2018). *Flask web development, one drop at a time*. Obtenido de http://flask.pocoo.org/

archive, I. (10 de 06 de 2018). *Internet archive wayback machine*. Obtenido de web archive: https://web.archive.org/

Foundation., P. S. (03 de 06 de 2018). *Python Software Foundation.* Obtenido de Python: https://www.python.org/doc/

gestionbasesdatos readthedocs. (20 de 07 de 2018). Obtenido de http://gestionbasesdatos.readthedocs.io/es/latest/Tema1/Teoria.html

Google. (03 de 06 de 2018). *Angular*. Obtenido de Angular: https://angular.io/

Google. (04 de 08 de 2018). *Angular*. Obtenido de Angular: https://angular.io/

Herranz Gómes, R. (Junio de 2014). Base de datos NoSQL: arquitectura y ejemplos de aplicación. Leganés: Universidad Carlos III de Madrid.

HotFrameworks. (s.f.). Obtenido de HotFrameworks: http://hotframeworks.com/

J. Garbage, M. (10 de 07 de 2018). *LIveedu.tv blog go on site*. Obtenido de LIveedu.tv blog: http://blog.liveedu.tv/django-flask-pyramid-framework-python/

Joyent. (03 de 06 de 2018). *Node.js Foundation*. Obtenido de Node.js: https://nodejs.org/es/

MongoDB. (08 de 08 de 2018). *MongoDB Documentation*. Obtenido de https://docs.mongodb.com/manual/

Mozilla. (20 de 07 de 2018). *MDN web docs*. Obtenido de MDN web docs: https://developer.mozilla.org/es/docs/Learn/Server-side/Django

Otto, M., & Fat. (03 de 06 de 2018). *Bootstrap*. Obtenido de Bootstrap: https://getbootstrap.com/

Pandorafms. (20 de 07 de 2018). Obtenido de https://blog.pandorafms.org/es/bases-de-datos-nosql/

Point, T. (03 de 06 de 2018). *Tutorials Point simply easy learning*. Obtenido de Tutorials Point: https://www.tutorialspoint.com/angular2/index.htm

Simulador de riego experto. (03 de 06 de 2018). Obtenido de SimulaReg: http://www.ies-sim.com/

Stonebraker, M. (2010). *SQL database v. NoSQL databases. Communications of the ACM.* Obtenido de https://www.devmedia.com.br/conheca-a-geracao-de-banco-de-dados-nosql-e-newsql/33202

Telefonica. (20 de 07 de 2018). *acens*. Obtenido de https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

Tutorials point. (20 de 07 de 2018). *Tutorials point*. Obtenido de Tutorials point: https://www.tutorialspoint.com/flask/flask_overview.htm

Twitter. (04 de 08 de 2018). *Bootstrap*. Obtenido de Bootstrap: https://getbootstrap.com/

Venners, B. (07 de 6 de 2018). *Artima scala consulting, training, books, and tools*. Obtenido de https://www.artima.com/intv/pythonP.html

You, E. (03 de 06 de 2018). *Vue.js*. Obtenido de Vue.js: https://vuejs.org/